UM1616
User manual

Trace infrastructure getting started

Introduction

The trace infrastructure provides a mechanism for capturing, decoding and displaying system trace data for SOCs that have embedded heterogeneous cores. It provides the user with a complete, easy to use trace environment. It includes the following features:

- ensures consistency and homogeneity for all core traces on a SoC
- works with any core or platform
- can be used to provide trace data for firmware stack, applications, inter-processor exchanges, operating systems, and so forth

The Trace Infrastructure supports STM2 and STM3 IPs associated with STPv1 and STPv2 data protocols respectively.

It is coupled to the Multi Target Trace API and data protocol

Contents

1 Definitions and acronyms 4

1.1 References 4

2 Trace infrastructure context 5

3 Trace infrastructure components 6

3.1 Hardware 6

3.2 Software package 6

4 Installing the trace infrastructure 7

4.1 System requirements 7

4.1.1 Linux environment 7

4.1.2 Windows environment 7

4.2 Hardware installation 7

4.3 Software installation 8

4.3.1 On Linux hosts 8

4.3.2 On Windows XP or Windows 7 8

5 How to initiate trace infrastructure 12

5.1 Global context (raw trace or MTT instrumented) 12

5.2 Preliminary to capturing trace 12

5.3 Capturing and viewing trace 13

5.3.1 Capturing trace 13

5.3.2 Viewing trace 14

6 Configuration and diagnostics 16

6.1 Configuring the test client 16

6.2 Executing the diagnostic test program 18

7 How to use trace infrastructure 19

7.1 STxP70 example 19

7.1.1 On an STiH415 board 19

7.1.2 Collect and display trace 21

7.2 ARM Cortex-A9 example 22
7.2.1 On ST-Ericsson MOP500/HREF500 board 22
7.2.2 Collect and display trace 24

8 Revision history 26

1 Definitions and acronyms

This chapter defines abbreviations and acronyms necessary to understand this document or that occur in other key process documents.

Table 1. Definitions and acronyms

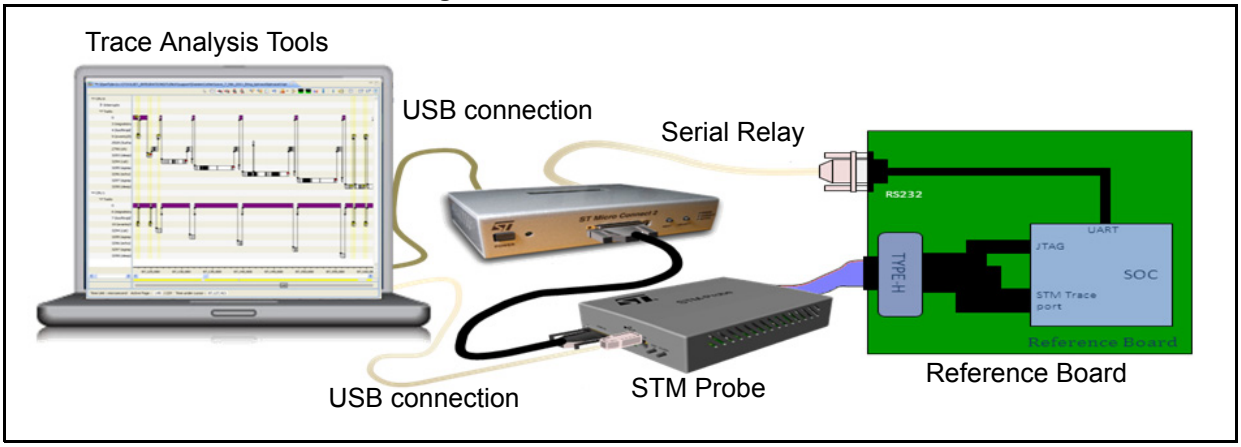
Term	Definition
LVDS	Low-voltage differential signalling.
Mb/s	Megabits per second.
MB/s	Megabytes per second.
MIPI	Mobile Industry Processor Interface. The MIPI Alliance is an open membership organization that includes leading companies in the mobile industry that share the objective of defining and promoting open specifications for interfaces in mobile terminals.
MTT	Multi-target trace.
PTI	Parallel trace Interface
RHEL	Red Hat Enterprise Linux
STMC2	ST Micro Connect 2 host-target interface.
SBC	Stand by controller.
STM	System trace module.
STP	System trace protocol.
System Trace	Messages generated by the software in order to give system (or application) level visibility

1.1 References

- *STM-Probe datasheet* (8282484)
- *Multi-Target Trace API user manual* (DocID023121)

2 Trace infrastructure context

Figure 1. STM-Probe connection



The Trace Infrastructure can be used either as a standalone toolset or interoperating with a core toolset. Whatever the context, the Trace Infrastructure is closely linked to the SoC including an STM IP.

3 Trace infrastructure components

The trace infrastructure consists of both hardware and software components. These are available separately.

3.1 Hardware

The trace infrastructure is based on the STMicroelectronics' system trace capture box called STM-Probe. To obtain the STM-Probe hardware, contact ST manufacturing. The product order code is "STM-PROBE". It consists of a trace box and associated LVDS cable with documentation.

3.2 Software package

The Trace Infrastructure software is available from ST FAEs.

It is available for Linux host RHEL5 and Fedora14/15 and Windows XP/ Windows 7 environments, it provides:

- libraries and drivers to enable trace capture
- host USB information (`libusb-win32` installer for Windows, USB rules files for Linux)
- a diagnostic program to validate the installation of the STM-Probe
- a trace server
- an MTT decoder and viewer
- a raw trace decoder and viewer
- a documentation portal

4 Installing the trace infrastructure

This chapter provides detailed instructions on how to install the trace infrastructure.

4.1 System requirements

Before attempting to install the trace infrastructure, make sure that the operating system meets the requirements described in this section.

4.1.1 Linux environment

Supported Linux environments are RHEL5 and Fedora 14 or 15. The trace infrastructure has been validated on 64-bit host PCs with a 32-bit emulation library. (64-bit machines are not supported: use the Trace Infrastructure on a 64-bit machine only if the 32-bit emulation library is installed.) In addition, check that the USB library is **libusb** v1.0; To ensure complete installation tools, you need root privileges to copy two files (specific rules) under specific directories.

4.1.2 Windows environment

Supported Windows environments are Windows XP and Windows 7. The Trace Infrastructure has been validated on Windows XP 32-bit with an X86 architecture. Check that the USB library installed on your PC is `libusb-win32-1.2.5.0`. The `libusb-win32` is a library that enables a user space application to access the USB devices in the Windows operating system. For more information, visit the project's web site at: `http://sourceforge.net/apps/trac/libusb-win32/wiki` where full documentation can be found.

If this library has not been installed, use the Windows installer to install it. You must have administrator privileges to complete this operation.

4.2 Hardware installation

The hardware installation can be completed in three steps:

1. Connect the STM-Probe (target) to the board.
2. Connect the STM-Probe to your PC (using USB).
3. Power up the STM-Probe *first* and then the board.

4.3 Software installation

4.3.1 On Linux hosts

To install the tools on Linux RHEL5 or Fedora 14 or 15, run the executable `ST_TraceInfrastucture_<version>.bin` and follow the instructions on the screen.

To complete the Linux installation, the following files (delivered in the directory `$STT/system/drivers/`) must be copied (with root privileges).

- Copy `$STT/system/drivers/99-stmprobe.rules` to the directory `/etc/udev/rules.d/`
- Copy `$STT/system/drivers/blacklist-stmprobe` to the directory `/etc/modprobe.d/`

The USB library v1.0 (if required) is in `$STT/system/libusb-1.0/` .

`$SST` is an environment variable that represents the installation path of the trace infrastructure software.

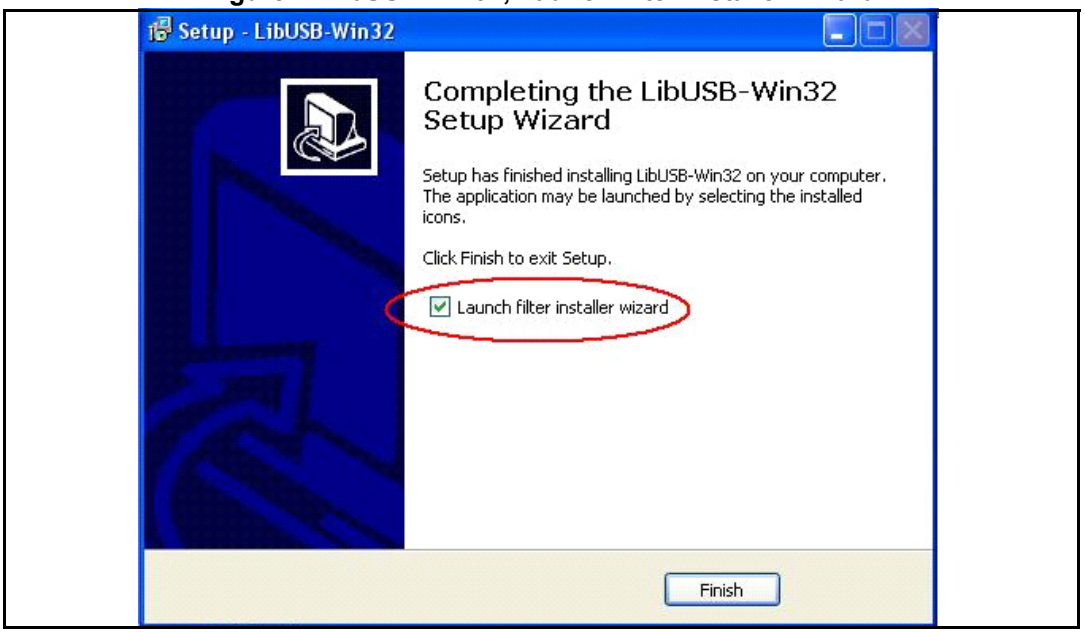
4.3.2 On Windows XP or Windows 7

To install the tools on Windows XP or Windows 7, double click on the Windows installer `Trace_Infrastructure_<version>.exe` and follow the instructions on the screen.

Before completing the installation, the Windows installer offers to set up the `libusb-win32-1.2.5.0` library on your PC. The trace infrastructure installer launches the `libusb-win32` setup wizard.

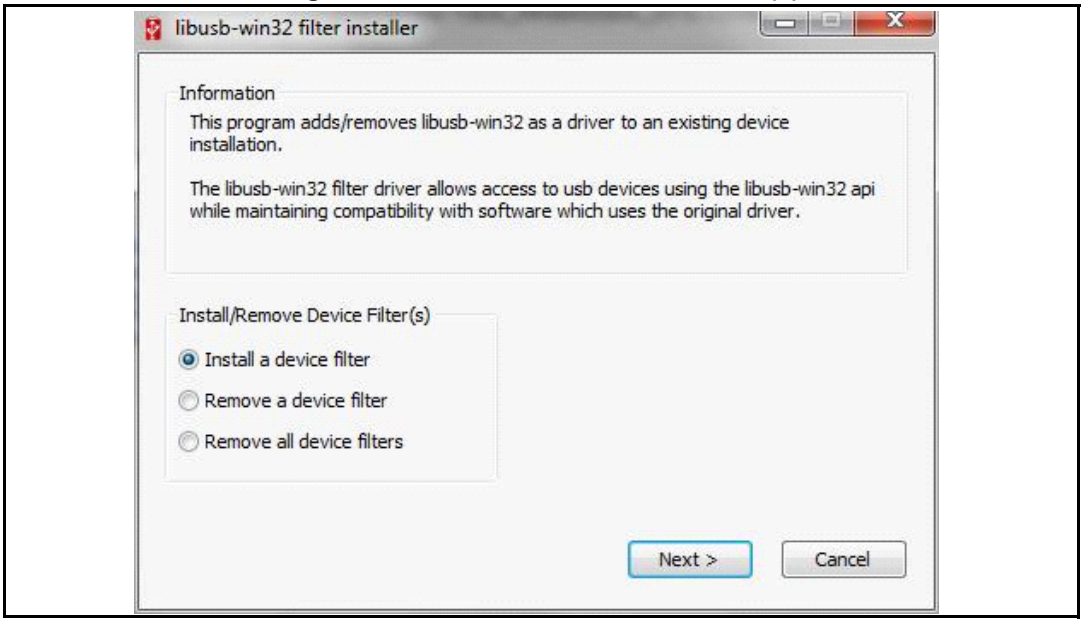
To set up the library, check the check box “Launch filter installer wizard”. (See *Figure 2*.)

Figure 2. LibUSB-win32, Launch filter installer wizard


Setup - LibUSB-Win32
Completing the LibUSB-Win32 Setup Wizard
Setup has finished installing LibUSB-Win32 on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
Launch filter installer wizard
Finish

Click on the **Finish** button to proceed to the next stage of the operation. The filter installer wizard displays the first screen (see *Figure 3*).

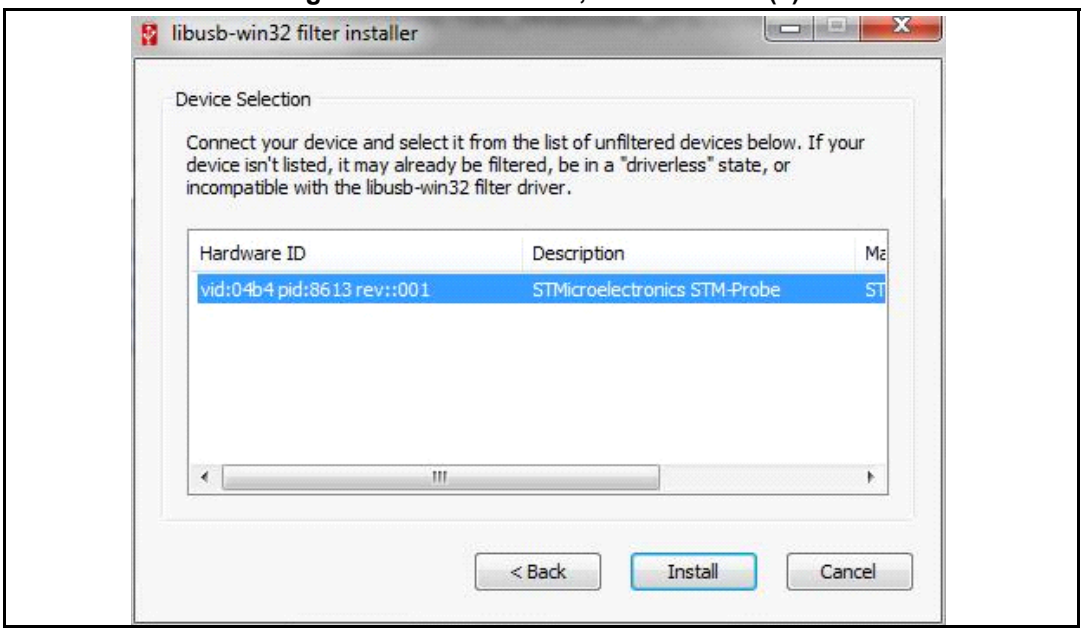
Figure 3. LibUSB-win32, filter installer (1)


libusb-win32 filter installer
Information
This program adds/removes libusb-win32 as a driver to an existing device installation.
The libusb-win32 filter driver allows access to usb devices using the libusb-win32 api while maintaining compatibility with software which uses the original driver.
Install/Remove Device Filter(s)
Install a device filter
Remove a device filter
Remove all device filters
Next >
Cancel

Select **Install a device filter** to set up the library. If no STM-Probe device is proposed in the Device Selection list, then the device is already installed. Click on **Cancel**.

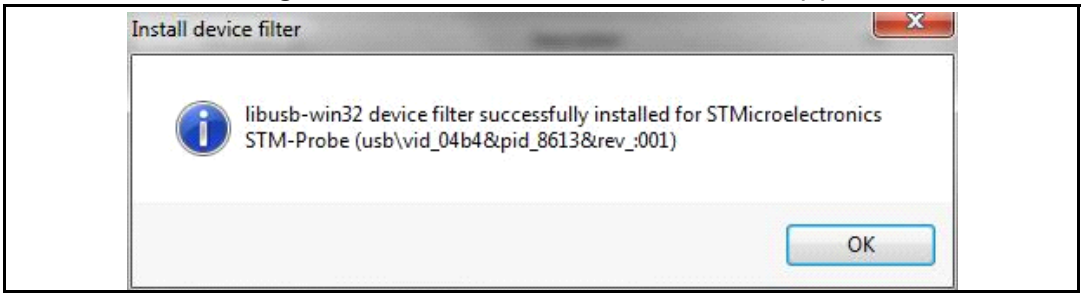
Otherwise, select the device with the following hardware identifier: `vib:04b4 pid:8613 rev::001` and click on the **Install** button (see *Figure 4*).

Figure 4. LibUSB-win32, filter installer (2)


libusb-win32 filter installer
Device Selection
Connect your device and select it from the list of unfiltered devices below. If your device isn't listed, it may already be filtered, be in a "driverless" state, or incompatible with the libusb-win32 filter driver.
Hardware ID
Description
vid:04b4 pid:8613 rev::001
STMicroelectronics STM-Probe
< Back
Install
Cancel

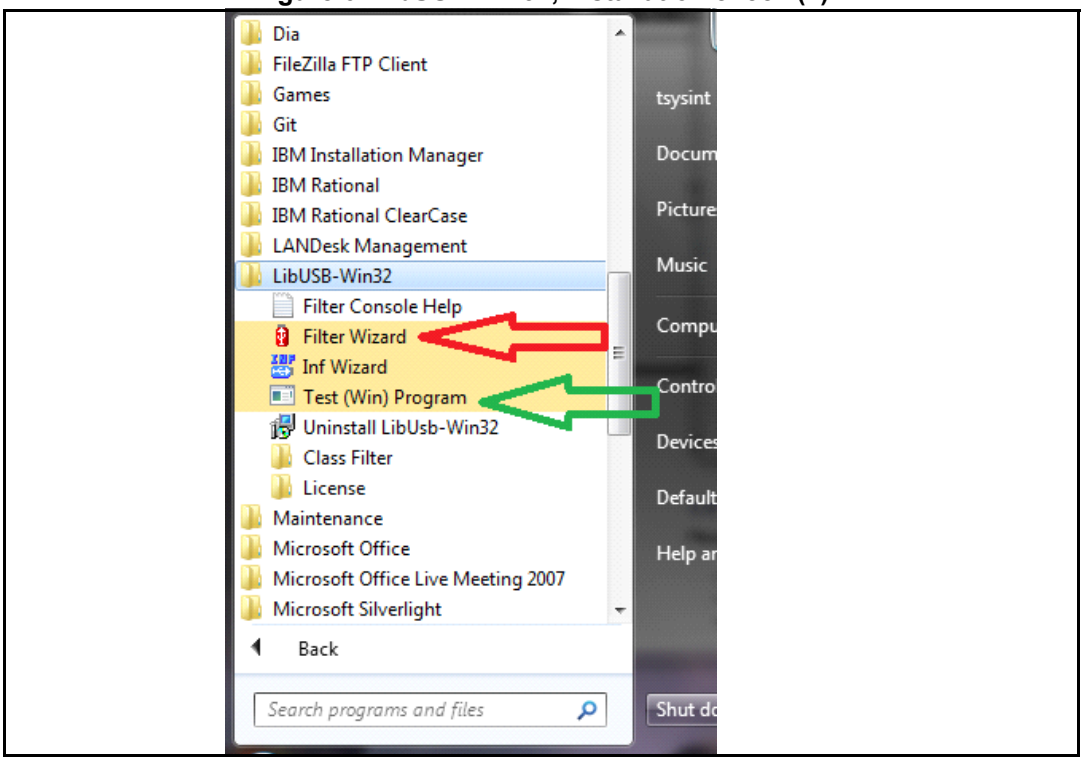
You can use the STM-Probe on your PC when the wizard displays the final installation message (see *Figure 5*).

Figure 5. LibUSB-win32, installation check (1)


Install device filter
libusb-win32 device filter successfully installed for STMicroelectronics STM-Probe (usb\vid_04b4&pid_8613&rev_:001)
OK

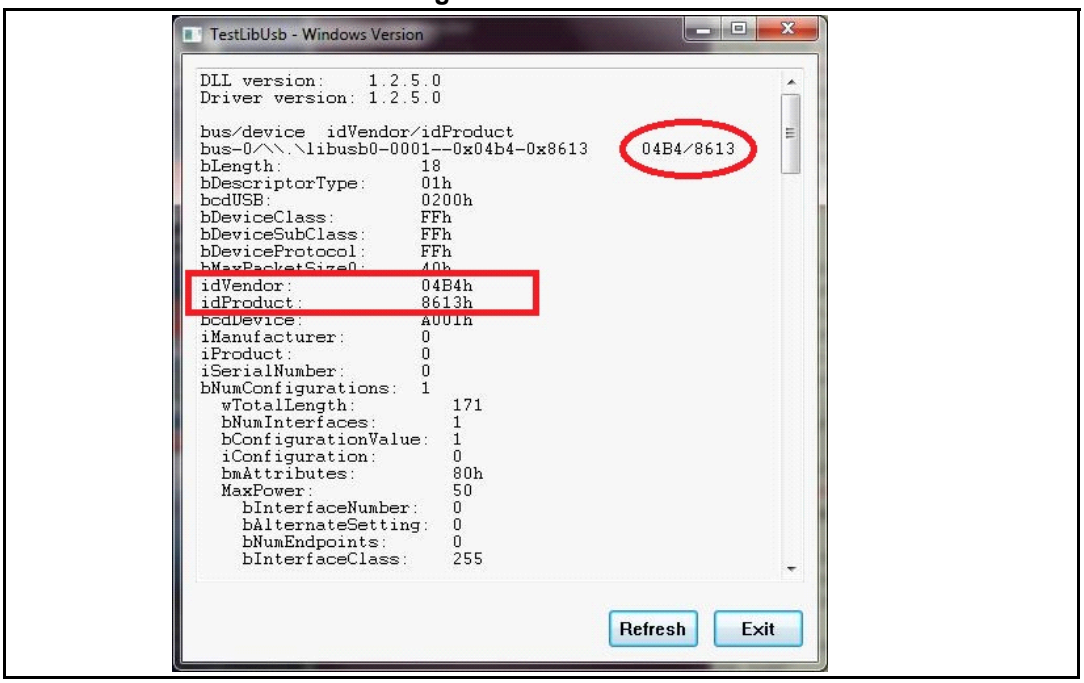
An alternative method for installing the STM-Probe device is to select **Filter Wizard** in the menu selection (as shown in *Figure 6*).

Figure 6. LibUSB-win32, installation check (2)



To test that the installation was completed successfully, select **Test (Win) Program** in the Start menu. This utility checks that one device have the values: `idVendor: 04B4` and `idProduct: 8613`. (See *Figure 7*.)

Figure 7. TestLibUsb



Note: Administrator rights are necessary to complete the installation. If you do not have these privileges, please contact your system administrator to complete the following actions:

- Install usb-win32 library if it is not yet installed on your machine:
  - Execute `<TOOLS_DIR>\system\liusb-win32\libusb-win32-devel-filter-1.2.5.0.exe` and follow the instruction on the screen
- Copy the STM-Probe drivers files:
  - Copy `<TOOLS_DIR>\system\drivers\x86\CyUSB.inf` to `c:\Windows\inf\`
  - Copy `<TOOLS_DIR>\system\drivers\x86\CyUSB.sys` to `c:\Windows\drivers\`

5 How to initiate trace infrastructure

5.1 Global context (raw trace or MTT instrumented)

An SoC environment (including STM IP) in which application code has been instrumented for the MTT libraries can use trace infrastructure. The MTT libraries follow MTT API and data protocol definitions and are available in core toolsets (such as STxP70) or STLinux.

5.2 Preliminary to capturing trace

Before starting to use the trace infrastructure, you must ensure the following tasks have been completed.

- The SoC has been initialized properly, especially with regards to system trace emission. Specifically, this means that:
  - GPIOs are configured to link STM output to the connector (see the appropriate SoC reference guide for more details)
  - the STM IP is configured to get trace from the application running on a core.

  The main information to be initialized is listed below. See the appropriate SoC reference guide for the correct initialization values. The addresses listed must be filled with correct values:

`STM_CONFIG_REGS_BASE`	This is the base address for the configuration register of the STM IP
`STM_CHANNELS_BASE`	This is the base address of the STM IP
`STM_CHANNELS_BASE_1`	Channel corresponding to a CPU
`STM_CHANNELS_BASE_2`	Channel corresponding to a CPU
`STM_CHANNELS_BASE_3`	Channel corresponding to a CPU
`STM_CR`	STM control register It is recommended for a first test to set the value to 0x1C0. This value corresponds to dividing the STM clock by 16, in order to have no performance issues. However another divider can be used.
`STM_MMC`	STM MIPI modes control register. This value corresponds to set the core as SW-Mode initiators
`STM_TER STM`	Trace enable register

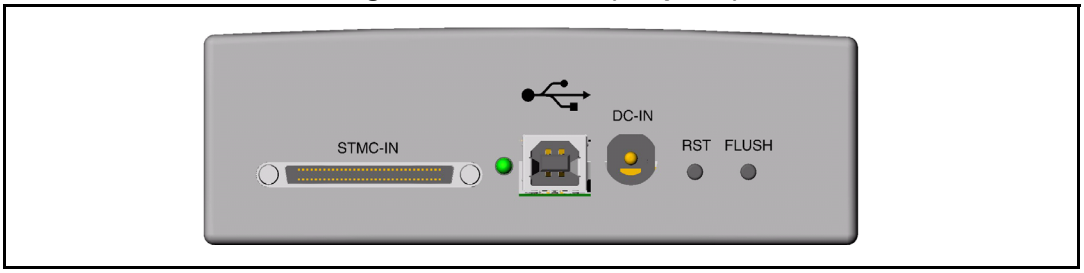
- The application code has been instrumented in order to emit trace through the STM IP. (See the appropriate core toolset user manual for more details.)

5.3 Capturing and viewing trace

5.3.1 Capturing trace

The LEDs on the STM-Probe panels provide STM-Probe status information. On the left panel of the STM-Probe, the green LED indicates that the USB connection is effective (see *Figure 8*).

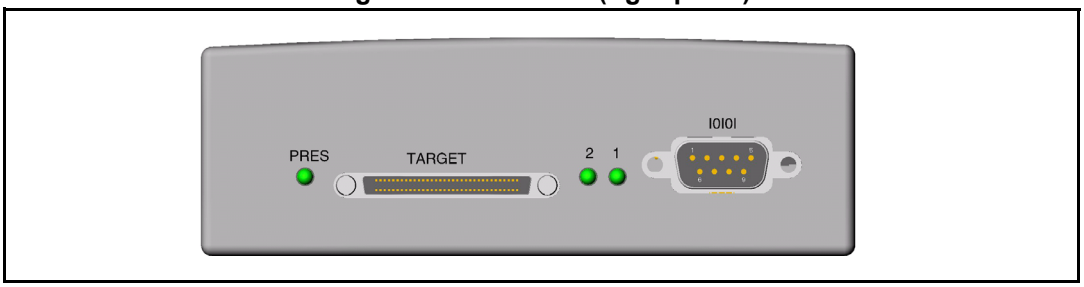
Figure 8. STM-Probe (left panel)



On the right panel of the STM-Probe, there are three LEDs (see *Figure 9*):

- the PRES LED is lit when the power is on
- LED 1 is lit while FPGA is loading, confirming STP trace input
- LED 2 is lit to indicate I^2C activity (host program activity)

Figure 9. STM-Probe (right panel)

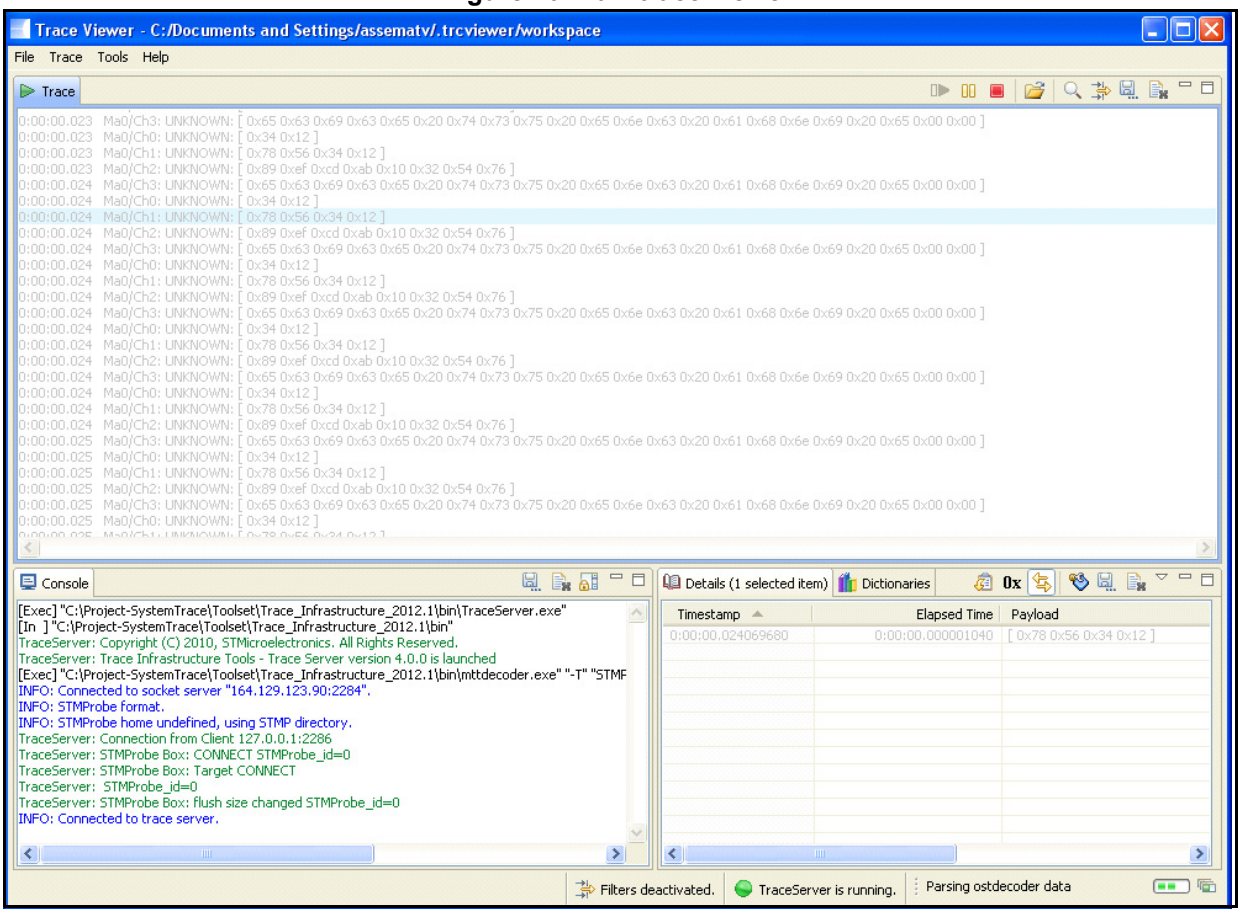


5.3.2 Viewing trace

The trace infrastructure provides two viewers:

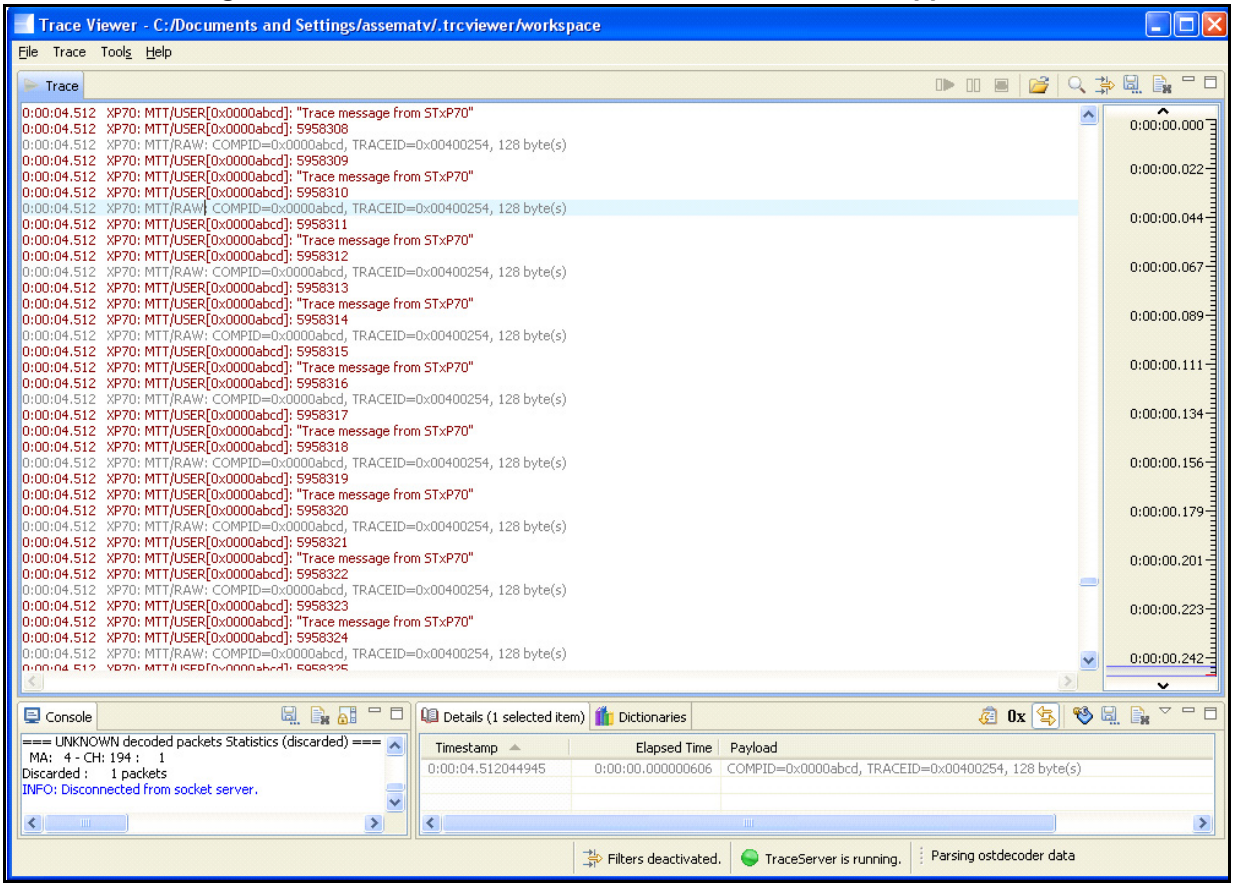
- a raw trace viewer that displays all traces (see *Figure 10*)

Figure 10. Raw trace viewer



- an MTT trace viewer that displays MTT instrumented trace only (refer to the appropriate core toolset user manual for details)

Figure 11. MTT instrumented trace viewer from STxP70 application


Trace Viewer - C:/Documents and Settings/assematv/.trcviewer/workspace
File Trace Tools Help
Trace
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958308
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958309
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958310
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958311
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958312
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958313
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958314
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958315
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958316
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958317
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958318
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958319
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958320
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958321
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958322
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958323
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958324
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:00.000
0:00:00.022
0:00:00.044
0:00:00.067
0:00:00.089
0:00:00.111
0:00:00.134
0:00:00.156
0:00:00.179
0:00:00.201
0:00:00.223
0:00:00.242
Console
=== UNKNOWN decoded packets Statistics (discarded) ===
MA: 4 - CH: 194 : 1
Discarded : 1 packets
INFO: Disconnected from socket server.
Details (1 selected item)
Dictionaries
0x
Timestamp
Elapsed Time
Payload
0:00:04.512044945
0:00:00.000000606
COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
Filters deactivated.
TraceServer is running.
Parsing ostdecoder data

6 Configuration and diagnostics

For information relating to the configuration of the test client, see *Section 6.1*.

After configuring the test client, use the STM-Probe diagnostic program provided with the package to discover if it can capture trace data. This program is described in *Section 6.2*. The test displays trace captured from the SoC and permits the capture of raw data provided by trace infrastructure.

6.1 Configuring the test client

The default configuration should satisfy the requirements of most use cases. However, depending upon your target, you may have to configure the tool by modifying the `Test_Stm-Probe_DLL.ini` file with target-specific parameters.

The following tables list the parameters that can be modified:

- for Ep8 configuration (flush), see *Table 2*
- for driver options, see *Table 3*
- for FPGA and driver configuration, see *Table 4*
- for debug and test options, see *Table 5*

Table 2. Ep8 configuration (flush)

Name	Value by default	Description
`ep8_read_size`	64*1024	Use this parameter to configure the buffer size for all bulk read on the ep8 (unit: octet). The value must be a multiple of 16.
`flush_size`	128*1024*1024	Use this parameter to configure the flush size (unit: octet). The value must be a multiple of 16.
`save_ep8_read_data`	`0`	Use this parameter to save all data that have been received from ep8. The trace infrastructure saves the data in the file defined by the `save_ep8_data_file_name` parameter.
`save_ep8_data_file_name`	`trace_ep8_out.dat`	Use this parameter to define the path of the file where data received from ep8 is to be saved.
`emul_ep8_data`	`0`	Use this parameter to load data from the file named by the `emul_ep8_data_file_name` parameter to ep8. No data is input from the FPGA. The trace infrastructure uses the driver layer only.
`emul_ep8_data_file_name`	`trace_ep8_in.dat`	Use this parameter to define the path of the file from where data for ep8 is to be loaded.

Table 3. Driver options

Name	Value by default	Description
stp_version	STPv1 = 1	Use this parameter to configure the driver and the FPGA with the appropriate protocol. Set this to: – 1 for STPv1 – 2 for STPv2.
disable_target_check	0	Use this parameter to disable the automatic check to establish if a target is connected at driver initialization.
nonstop_flush	0 (disabled)	Set this parameter to 1 to disable runtime mode and configure the driver to use flush mode continuously. In this mode, the trace infrastructure does not use a filter and receives all data irrespective of the channel.
select_master	0	Use this parameter to configure in the FPGA which master send messages.
disable_flush_sync	0	Set this parameter to 1 to configure the driver to not wait for an ASYNC message following a request to flush (except for the first request).
default_channel	DEFAULT_CHANNEL = 10	Use this parameter to set the default channel. Trace infrastructure uses the default channel when it has not yet received a channel message.
ring_buffer_no_runtime	0 (disabled)	Set this parameter to 1 to configure the driver not to store a runtime message in the ring buffer. This means when we do a flush we have no runtime information.

Table 4. FPGA and drivers configuration

Name	Value by default	Description
always_load_FPGA	0	Set this parameter to 1 to load the FPGA every time the driver is initialized, irrespective of FPGA version. (By default, the trace infrastructure loads the FPGA only when the version is different than the one it requires.)
rbf_file_name	logplug.rbf	This parameter provides the path of the FPGA file. The parameter is set by the driver layer. Users have no need to change it, because the driver and FPGA are delivered in the same package.
hex_file_name	fx2.hex	This parameter provides the path of the hex file. The parameter is set by the driver layer. Users have no need to change it, because the driver and FPGA are delivered in the same package.
rbf_file_version	RBF_FILE_VERSION	This parameter shows the FPGA version number. The parameter is set by the driver layer. Users have no need to change it, because the driver and FPGA are delivered in the same package.

Table 5. Debug and test options

Name	Value by default	Description
`internal_FPGA_test`	`0` (disabled)	Set this parameter to 1 to generate STP messages from the FPGA without using a target. This option can generate a large amount of messages without a break. It works only for STPv1.
`debug_level`	`1`	Use this parameter to select the required debug level. The trace infrastructure prints a message only when its level is less than the level set here. The permitted values are: – 0: no log – 1: errors only – 2: level 1 plus initialization statistics – 3: level 2 plus timeout and driver overflow – 4: level 3 plus decoded messages (this level suffers from degraded performance)
`print_version`	`0`	Set this parameter to print information concerning the FPGA firmware version and DDR2 version.
`request_dump_on_error`	`0` (disabled)	Use this option to activate a dump after an opcode error in order to locate the error. The output is stored in a `.dat` file.

6.2 Executing the diagnostic test program

To check if everything has been set up correctly, run the diagnostic program `Test_Stm-Probe_DLL.exe`. The test is a short example of instrumented code and provides raw display of the trace captured.

7 How to use trace infrastructure

7.1 STxP70 example

7.1.1 On an STiH415 board

This section describes the steps required to configure an STiH415 board to collect traces from an application running on the STxP70 core. Traces are collected using the STM-Probe acquisition box. This example is specific to the standby controller (SBC). The STxP70 application instrumentation is based on the MTT API function.

Environment and application setup

Before starting, install the following:

- the trace infrastructure toolset
- the STxP70 toolset version 2012.1 or later (for information on how to obtain the toolset, contact stxp70.support@st.com)
- the ST Micro Connection Package and associated ST TargetPack supporting the STxP70 toolset (STMicroelectronics recommend that you install the latest available version of the ST Micro Connection Package)

Then, setup the environment:

- The first step is to setup the STxP70 toolset environment. Please refer to STxP70 toolset documentation. Then initialize, according to the host platform, all the necessary environment variables needed by target pack.

`STMC_DIR`	ST Micro Connection Package installation directory
`TARGETPACK_PATH`	TARGET PACK installation directory
`STMC2_IP_ADDR`	STMC2 IP Address
`STTP_XML_PATH`	sttp.xml file directory

- According to the host platform, update your `PATH` environment variable:

`PATH`	`$STMC_DIR/bin`
`LD_LIBRARY_PATH`	`$STMC_DIR/lib`

Then, build your application:

- Build your application with the STxP70 compiler and link with the STxP70 MTT library. Please refer to MTT Library Documentation.

Finally, launch the STxP70 debugger:

- To connect to debugger, you must declare a suitable target in your `target.ini` file. *Figure 12* shows an example:

Figure 12. Example target.ini file

```
[xp70_lpm]
Rule=target:type EMU
Rule=define V4
Section=Core
Section=Common
Section=Memory_Map_lpm OwnerParam=<BEGIN>
connect STMC2 -retry 2 100 -p "no_reset=1" -timeout 3000 -no_jtag_test
-target_pack "${STMC2_IP_ADDR}:b2000stxh415:xp70_lpm no_convertor_abort=1
silent=1 tck_frequency=10000000 targetpack_debug=jtag
stmc_pokepeek_timeout=10000 module_function=stm:cpt_table_orly"
<OWNER PARAM END>

[Memory_Map_lpm]
Rule=require EMU && (V4)
MemoryMap = DRAM 0x00000000 0x00002000 rw
MemoryMap = PRAM 0x00400000 0x00008000 rwx
MemoryMap = ERAM 0x00800000 0x02000000 rwx
MemoryMap = ERAM 0xA00000 0xFF600000 rwx
```

Note: The `stm:cpt_table_orly` tokens that appear in the `connect` command are Python scripts in charge of respectively initializing the STM and the MTT API. You must be sure that these two files are in your target pack directory.

The command line to launch the STxP70 debugger is the following:

```
sxgdb --targetfile=<your target.ini>
-prop=connect_cmd=st.config.fetch_enable.on --target=<your target> <your
instrumented application>
```

As an example:

```
sxgdb --targetfile=target.ini -prop=connect_cmd=st.config.fetch_enable.on
--target=xp70_lpm exampleMTTAPI.out
```

If you use STWorkbench, add `-prop=connect_cmd=st.config.fetch_enable.on` in the **More options** field of the debug configuration (**Debug Configurations** > **Tab Debugger** > **Main** > **More options**).

7.1.2 Collect and display trace

Before running the application, launch the trace viewer (named **trcviewer**) to connect to STM-Probe and display trace data. If you start the application before launching the decoder you may lose some trace data.

Figure 13. Example of trace viewer initialization (1)

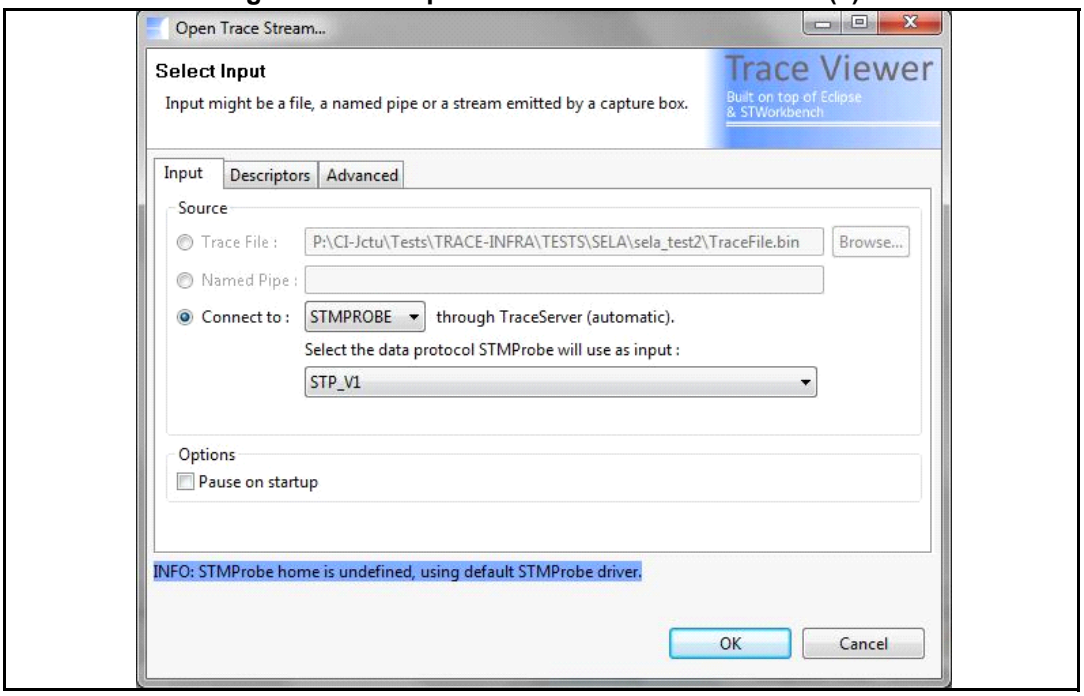
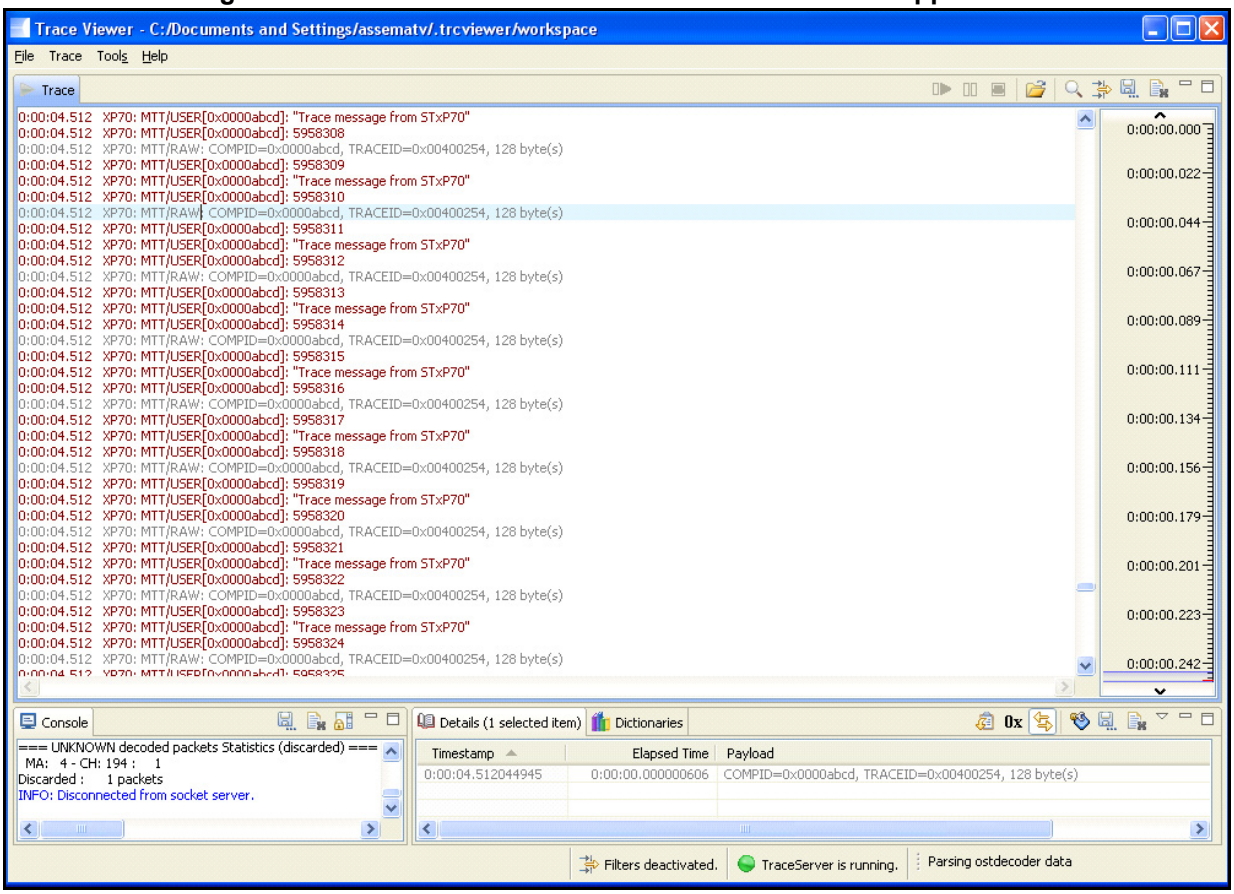


Figure 14. MTT instrumented trace viewer from STxP70 application


Trace Viewer - C:/Documents and Settings/assematv/.trcviewer/workspace
File Trace Tools Help
Trace
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958308
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958309
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958310
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958311
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958312
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958313
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958314
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958315
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958316
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958317
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958318
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958319
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958320
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958321
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958322
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958323
0:00:04.512 XP70: MTT/USER[0x0000abcd]: "Trace message from STxP70"
0:00:04.512 XP70: MTT/USER[0x0000abcd]: 5958324
0:00:04.512 XP70: MTT/RAW: COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
0:00:00.000
0:00:00.022
0:00:00.044
0:00:00.067
0:00:00.089
0:00:00.111
0:00:00.134
0:00:00.156
0:00:00.179
0:00:00.201
0:00:00.223
0:00:00.242
Console
=== UNKNOWN decoded packets Statistics (discarded) ===
MA: 4 - CH: 194 : 1
Discarded : 1 packets
INFO: Disconnected from socket server.
Details (1 selected item)
Dictionaries
0x
Timestamp
Elapsed Time
Payload
0:00:04.512044945
0:00:00.000000606
COMPID=0x0000abcd, TRACEID=0x00400254, 128 byte(s)
Filters deactivated.
TraceServer is running.
Parsing ostdecoder data

7.2 ARM Cortex-A9 example

7.2.1 On ST-Ericsson MOP500/HREF500 board

This section describes the steps on the MOP500/HREF500 board to collect traces from an application running on ARM Cortex-A9. Traces are collected using the STM-Probe system trace capture box.

Environment and application setup

Before starting, you need to install:

- the trace infrastructure toolset
- an ARM Cortex-A9 debugger
- ST Micro Connection Package (STMicroelectronics recommend that you install the latest available version)

Next, setup the environment:

1. The first step is to setup the ARM Cortex-A9 environment.
2. Initialize, according to the platform, all the necessary environment variables needed by ST Micro Connection Package:
   `LD_LIBRARY_PATH=<installation directory>/lib`

Then instrument the application.

Note: The application must initialize the STM and must be instrumented to emit trace over STM. Refer to the STM documentation of your board to be able to configure the STM.

Figure 15. Example page 1

```
#define STM_CHANNELS_BASE 0x80100000 // STM Channel Base Address
#define STM_CONFIG_REGS_BASE 0x8010F000 // STM Configuration Register Base Address

static unsigned int volatile *PRCM_GPIOCR = (unsigned int volatile *) 0x80157138;
static unsigned int volatile *GPIO0_AFSLA = (unsigned int volatile *) 0x8000E120;
static unsigned int volatile *GPIO0_AFSLB = (unsigned int volatile *) 0x8000E124;

// Get the STM IP ready to use
static unsigned int volatile *STM_CR = (unsigned int volatile *)
(STM_CONFIG_REGS_BASE);
static unsigned int volatile *STM_MMC = (unsigned int volatile *)
(STM_CONFIG_REGS_BASE + 0x08);
static unsigned int volatile *STM_TER = (unsigned int volatile *)
(STM_CONFIG_REGS_BASE + 0x10);
static unsigned int volatile *STM_TDSR = (unsigned int volatile *)
(STM_CONFIG_REGS_BASE + 0x18);
static unsigned int volatile *STM_TFSR = (unsigned int volatile *)
(STM_CONFIG_REGS_BASE + 0x28);

typedef volatile struct {
unsigned long long no_stamp;
unsigned long long stamp;
} s_stm_channel;

// it is up to the calling application to provide a definition for stm_channels
volatile s_stm_channel * const stm_channels = (s_stm_channel *) STM_CHANNELS_BASE;

// initialize the STM IP

// GPIO[70..74]
*GPIO0_AFSLA |= 0x000007C0;
*GPIO0_AFSLB |= 0x000007C0;

*PRCM_GPIOCR &= ~(1 << 11); // get Modem STM traces out of GPIO[70..74]

// set cores from 0 to 4 as SW-mode initiators
*STM_MMC &= 0x20;

// set PRCMU as HW-mode
*STM_MMC |= 0x20;

// clock divisor: we chose XCKDIV=11b for ED (XCKDIV=00b for palladium)
*STM_CR |= 0x000000C0;

// allow all masters to write onto the STM
*STM_TER |= 0x01; // A9_0 & A9_1

... ...
```

Figure 16. Example page 2

```
// Emit Trace through STM

// Emit a D16 STP message on channel 0:
*((volatile unsigned short *) &(stm_channels[0].no_stamp)) = 0x1234;

// Emit a D16 timestamped STP message on channel 0:
*((volatile unsigned short *) &(stm_channels[0].stamp)) = 0x1234;
// Emit a D32 STP message on channel 1: *((volatile unsigned int *)
&(stm_channels[1].no_stamp)) = 0x12345678;

// Emit a D32 timestamped STP message on channel 1:
*((volatile unsigned int *) &(stm_channels[1].stamp)) = 0x12345678;

// Emit a D64 STP message on channel 2:
*((volatile unsigned long long *) &(stm_channels[2].no_stamp)) =
0x76543210ABCDEF89ULL;

// Emit a D64 Timestamped STP message on channel 2:
*((volatile unsigned long long *) &(stm_channels[2].stamp)) = 0x76543210ABCDEF89ULL;
```

Then launch the ARM Cortex-A9 debugger.

7.2.2 Collect and display trace

Before running the application, launch the trace viewer (**trcviewer**) and select the option `--display-unknown` (raw display) to connect to STM-Probe and display traces.

If you start the application before launching the viewer you may lose some trace data.

Figure 17. Example of trace viewer initialization (2)

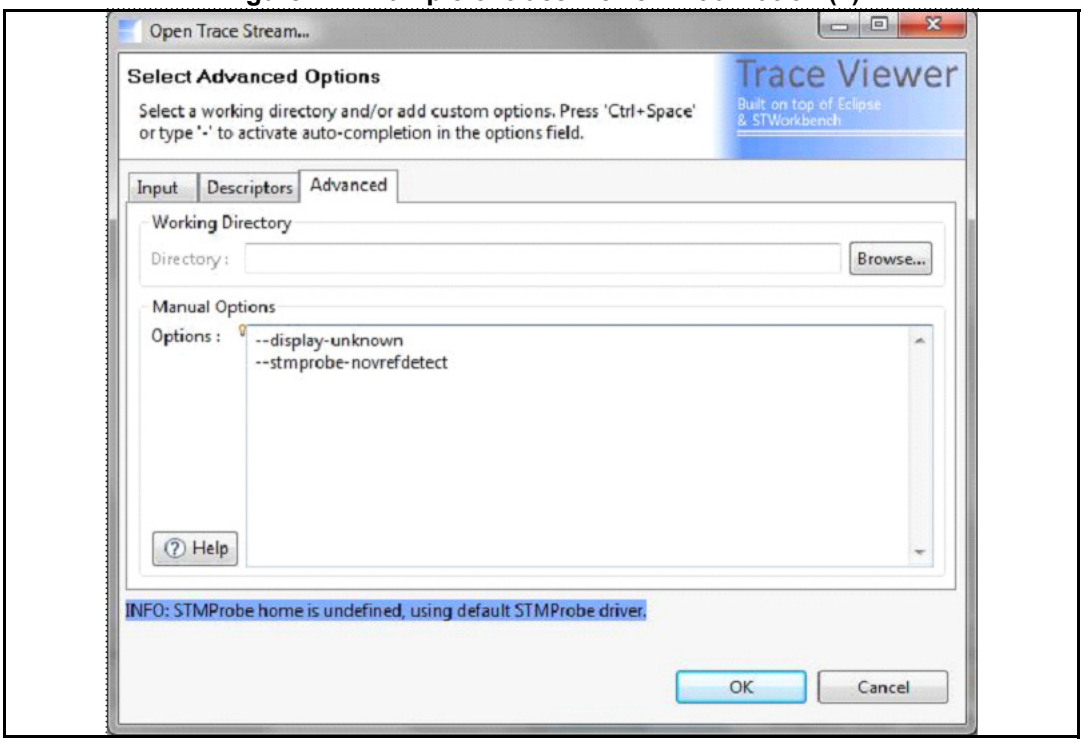
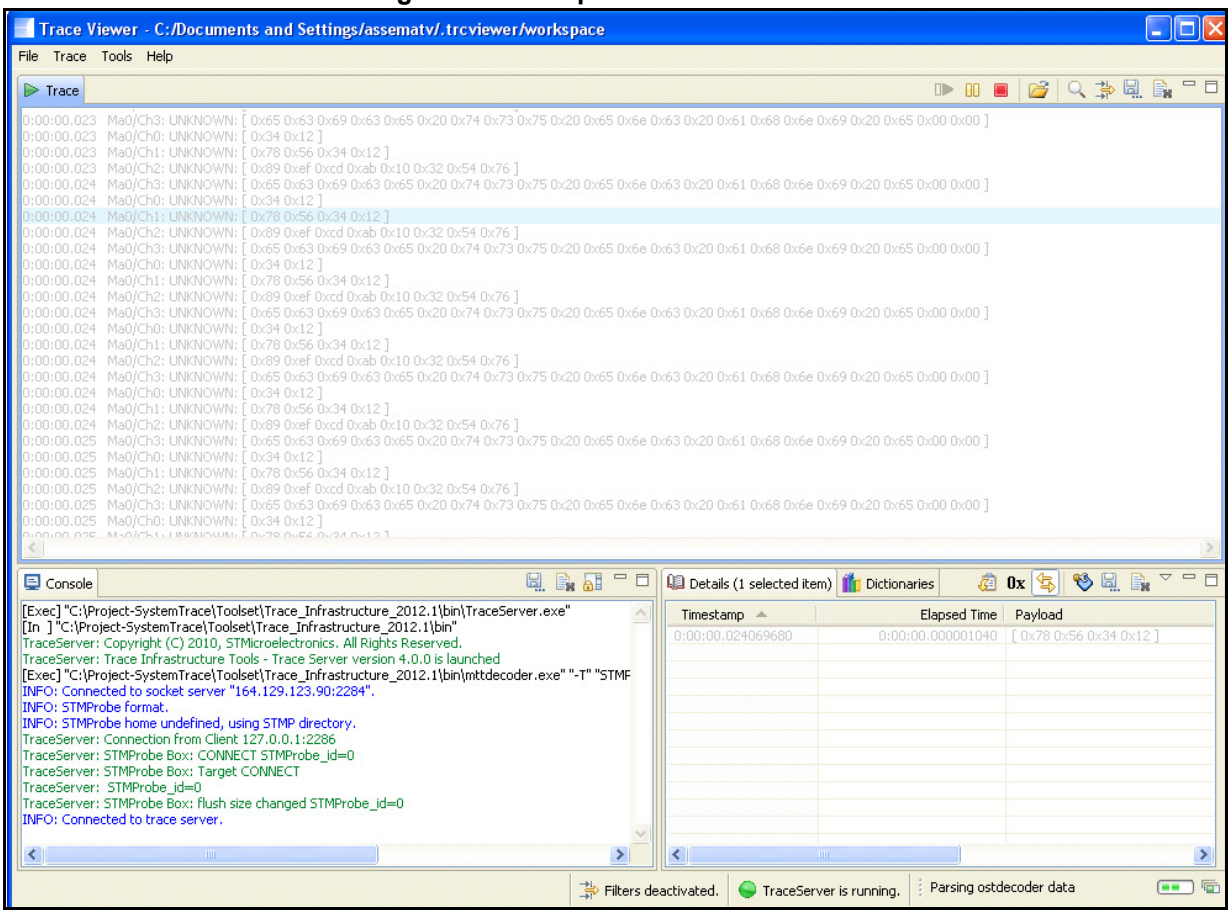


Figure 18. Example of raw trace viewer



8 Revision history

Table 6. Document revision history

Date	Revision	Changes
19-Mar-2013	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT AUTHORIZED FOR USE IN WEAPONS. NOR ARE ST PRODUCTS DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com